## Контрольная работа

### Тема: Последовательные контейнеры STL и модульное тестирование

Цель: Сформировать практические навыки разработки абстракций данных на основе контейнеров STL и модульного тестирования средствами Visual Studio.

### Задание

Реализовать обработку данных пользовательского типа (объектов класса) с помощью контейнера в соответствии с вариантом задания и со следующей спецификацией:

* приложение заполняет контейнер данными, которые вводятся пользователем с консоли;
* выводит содержимое контейнера на консоль для контроля ввода;
* выполняет сортировку контейнера в порядке возрастания значений объектов с помощью алгоритма или метода контейнера;
* выводит содержимое контейнера на консоль для контроля операции;
* выполняет сортировку контейнера в порядке убывания значений объектов с помощью алгоритма или метода контейнера;
* выводит содержимое контейнера на консоль для контроля операции;
* вычисляет сумму значений объектов с помощью алгоритма и выводит значение на консоль.

Протестировать методы класса объектов, помещаемых в контейнер, используя средства модульного тестирования Visual Studio. Тестовые наборы необходимо построить на основе критериев тестирования C0,C1,C2 в зависимости от варианта задания.

### Варианты задания

Варианты заданий контрольной работы представлены в таблице ниже. Вариант определяет тип контейнера используемого для обработки данных, класс объектов помещаемых в контейнер, критерии тестирования разработанного приложения.

|  |  |  |  |
| --- | --- | --- | --- |
| №Варианта | Тип контейнера | Класс объектов | Критерий тестирования |
| 1 | deque | Простая дробь | С0,C1 |
| 2 | deque | Комплексное число | С0,C1 |
| 3 | deque | Р-ичное число | С0,C1 |
| 4 | deque | Простая дробь | C1,C2 |
| 5 | deque | Комплексное число | C1,C2 |
| 6 | deque | Р-ичное число | C1,C2 |
| 7 | vector | Простая дробь | С0,C1 |
| 8 | vector | Комплексное число | С0,C1 |
| 9 | vector | Р-ичное число | С0,C1 |
| 10 | vector | Простая дробь | C1,C2 |
| 11 | vector | Комплексное число | C1,C2 |
| 12 | vector | Р-ичное число | C1,C2 |
| 13 | list | Простая дробь | С0,C1 |
| 14 | list | Комплексное число | С0,C1 |
| 15 | list | Р-ичное число | С0,C1 |
| 16 | list | Простая дробь | C1,C2 |
| 17 | list | Комплексное число | C1,C2 |
| 18 | list | Р-ичное число | C1,C2 |

### Рекомендации к выполнению

1. Заданную обработку данных реализуйте как консольное приложение, используя классы языка программирования и библиотеку шаблонов STL.
2. Добавьте в класс объектов (в соответствии с вариантом задания), помещаемых в контейнер перегруженные операторы < и > для выполнения сортировки. В контрольной работе используются классы, разработанные вами в лабораторных работах.
3. Выполняйте сортировку контейнера с помощью алгоритма sort или метода контейнера в зависимости от варианта.
4. Вычисление суммы значений объектов контейнера выполняйте с помощью алгоритма accumulate и функционального объекта plus<A>(), здесь А – класс ваших объектов;
5. Для выполнения описанных в задании операций по обработке данных, разработайте класс со следующим описанием:

class InOutDo

{

public:

 static void Input(…)

 {

 //Вводим данные с клавиатуры и заносим в контейнер.

 }

 static void Output(…)

 {

//Выводим содержимое контейнера на монитор.

}

 static A Sum(…)

 {

 //Находим сумму

 }

 static void SortUp(…)

 {

 //Сортируем вектор по возрастанию

 }

 static void SortDown(…)

 {

 //Сортируем вектор по убыванию

 }

};

Тогда функция main примет примерно такой вид:

int \_tmain(int argc, \_TCHAR\* argv[])

{

 SetConsoleCP(1251);

 SetConsoleOutputCP(1251);

 //Описание вашего Контейнера с объектами вашего класса А.

 //Контейнер<A> m;

 InOutDo::Input(m);

 InOutDo::Output(m);

 InOutDo::SortUp(m);

 InOutDo::Output(m);

 InOutDo::SortDown(m);

 InOutDo::Output(m);

 cout << InOutDo::Sum(m).get() << endl;

 system("PAUSE");

 return 0;

}

Примеры программ обработки данных с помощью деки, вектора и списка приведены ниже.

**Пример 1. Обработка данных с декой.**

// ConsoleDeque.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include <iostream>

#include <string>

#include <sstream>

#include <deque>

#include <algorithm>

#include <numeric>//Определяет шаблонны функций контейнера, которые выполняют алгоритмы числовой обработки.

#include <functional>

#include "windows.h"

using namespace std;

//Класс объектов, помещаемых в контейнер.

class A

{

 int n, d;

public:

 A(int n = 0, int d = 1) : n(n), d(d){};

 A operator+(const A b)const

 {

 return A((n\*b.d + b.n\*d), d\*b.d);

 };

 string get()

 {

 string a;

 ostringstream os;

 os << n << "/" << d;

 return os.str();

 };

 bool operator>(const A b)const

 {

 return n\*b.d > d\*b.n;

 };

 bool operator<(const A b)const

 {

 return n\*b.d < d\*b.n;

 };

};

int \_tmain(int argc, \_TCHAR\* argv[])

{

 SetConsoleCP(1251);

 SetConsoleOutputCP(1251);

 //Дека простых дробей

 deque<A> m;

 //Итератор для деки

 deque<A>::iterator iter;

 //Проталкиваем дроби в деку

 m.push\_back(A(2));

 m.push\_back(A(3));

 m.push\_back(A(1));

 m.push\_back(A(5));

 m.push\_back(A(9));

 m.push\_back(A(7));

 //Сортируем деку по возрастанию

 sort(m.begin(), m.end(), less<A>());

 for (iter = m.begin(); iter != m.end(); iter++)

 cout << iter->get() << " ";

 cout << endl;

 //Сортируем деку по убыванию

 sort(m.begin(), m.end(), greater<A>());

 for (int j = 0; j != m.size(); j++)

 cout << m[j].get() << " ";

 cout << endl;

 //Находим сумму

 A sum = accumulate(m.begin(), m.end(), A(), plus<A>());

 cout << sum.get() << endl;

 system("PAUSE");

 return 0;

 return 0;

}

**Пример 2. Обработка данных с вектором.**

// ConsoleVector.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include <iostream>

#include <string>

#include <sstream>

#include <vector>

#include <algorithm>

#include <numeric>//Определяет шаблонны функций контейнера, которые выполняют алгоритмы числовой обработки.

#include <functional>

#include "windows.h"

using namespace std;

//Класс объектов, помещаемых в контейнер.

class A

{

 int n, d;

public:

 A(int n = 0, int d = 1) : n(n), d(d){};

 A operator+(const A b)const

 {

 return A((n\*b.d + b.n\*d), d\*b.d);

 };

 string get()

 {

 string a;

 ostringstream os;

 os << n << "/" << d;

 return os.str();

 };

 bool operator>(const A b)const

 {

 return n\*b.d > d\*b.n;

 };

 bool operator<(const A b)const

 {

 return n\*b.d < d\*b.n;

 };

};

int \_tmain(int argc, \_TCHAR\* argv[])

{

 SetConsoleCP(1251);

 SetConsoleOutputCP(1251);

 //Вектор простых дробей

 vector<A> m;

 //Итератор для вектора

 vector<A>::iterator iter;

 //Проталкиваем дроби в вектор

 m.push\_back(A(2));

 m.push\_back(A(3));

 m.push\_back(A(1));

 m.push\_back(A(5));

 m.push\_back(A(9));

 m.push\_back(A(7));

 //Сортируем вектор по возрастанию

 sort(m.begin(), m.end(), less<A>());

 for (iter = m.begin(); iter != m.end(); iter++)

 cout << iter->get() << " ";

 cout << endl;

 //Сортируем вектор по убыванию

 sort(m.begin(), m.end(), greater<A>());

 for (int j = 0; j != m.size(); j++)

 cout << m[j].get() << " ";

 cout << endl;

 //Находим сумму

 A sum = accumulate(m.begin(), m.end(), A(), plus<A>());

 cout << sum.get() << endl;

 system("PAUSE");

 return 0;

}

**Пример 3. Обработка данных со списком.**

// ConsoleList.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include <iostream>

#include <string>

#include <sstream>

#include <list>

#include <algorithm>

#include <numeric>//Определяет шаблонны функций контейнера, которые выполняют алгоритмы числовой обработки.

#include <functional>

#include "windows.h"

using namespace std;

//Класс объектов, помещаемых в контейнер.

class A

{

 int n, d;

public:

 A(int n = 0, int d = 1) : n(n), d(d){};

 A operator+(const A b)const

 {

 return A((n\*b.d + b.n\*d), d\*b.d);

 };

 string get()

 {

 string a;

 ostringstream os;

 os << n << "/" << d;

 return os.str();

 };

 bool operator>(const A b)const

 {

 return n\*b.d > d\*b.n;

 };

 bool operator<(const A b)const

 {

 return n\*b.d < d\*b.n;

 };

};

int \_tmain(int argc, \_TCHAR\* argv[])

{

 SetConsoleCP(1251);

 SetConsoleOutputCP(1251);

 //Список простых дробей

 list<A> m;

 //Итератор для списка

 list<A>::iterator iter;

 //Проталкиваем дроби в вектор

 m.push\_back(A(2));

 m.push\_back(A(3));

 m.push\_back(A(1));

 m.push\_back(A(5));

 m.push\_back(A(9));

 m.push\_back(A(7));

 //Сортируем список по возрастанию

 //sort(m.begin(), m.end(), );

 m.sort(less<A>());

 for (iter = m.begin(); iter != m.end(); iter++)

 cout << iter->get() << " ";

 cout << endl;

 //Сортируем список по убыванию

 //sort(m.begin(), m.end(), );

 m.sort(greater<A>());

 for (iter = m.begin(); iter != m.end(); iter++)

 cout << iter->get() << " ";

 cout << endl;

 //Находим сумму

 A sum = accumulate(m.begin(), m.end(), A(), plus<A>());

 cout << sum.get() << endl;

 system("PAUSE");

 return 0;

 return 0;

}

**Пример модульного теста для тестирования перегруженных операторов +, == класса А приведён ниже.**

#include "stdafx.h"

#include "CppUnitTest.h"

#include "..\ConsoleDeque\A.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1

{

 TEST\_CLASS(UnitTest1)

 {

 public:

 TEST\_METHOD(Add)

 {

 A a = A(1, 2);

 A b = A(1, 2);

 A c = a + b;

 string s = "4/4";

 Assert::AreEqual(s, c.get());

 }

 TEST\_METHOD(Equal)

 {

 A a = A(1, 2);

 A b = A(2, 4);

 A c = a + b;

 string s = "4/4";

 Assert::AreEqual(true, a == b);

 }

 };

}

### Порядок выполнения

Реализуйте задание с помощью проекта «консольное приложение»:

1. Создайте консольное приложение и сохраните его под именем CJob\_1.
2. Добавьте к исходному тексту консольного приложения описание вашего класса (объекты которого вы будете обрабатывать с помощью контейнера) в соответствии с вариантом задания.
3. Добавьте в описание класса перегруженные операторы отношения (<,>) и, если необходимо, недостающие методы.
4. Добавьте в ваше решение класс модульного теста C++. Разработайте тестовый набор данных для тестирования методов вашего класса по критериям тестирования в соответствии с вариантом и протестируйте их, используя средства модульного тестирования Visual Studio.
5. Добавьте к исходному тексту консольного приложения описание класса InOutDo в соответствии с заданием.
6. Используя методы класса InOutDo, выполните требуемую обработку данных.

Таблица. Тестовый набор для тестирования методов класса объектов, помещаемых в контейнер.

|  |
| --- |
| Тестовый набор для тестирования операции Сложить для класса простых дробей |
| Номер теста | Исходные данные | Ожидаемый результат |
| Вход | Вход | Возвращаемое значение |
| 1 | (1/2) | (1/2) | (4/4) |

### Контрольные вопросы

1. В чём состоит сущность критерия C0?
2. В чём состоит сущность критерия C1?
3. В чём состоит сущность критерия C2?
4. Что такое УПГ?
5. Что такое путь в УПГ?
6. Что такое ветвь УПГ?
7. В каком файле описан последовательный контейнер deque?
8. В каком файле описан последовательный контейнер vector?
9. В каком файле описан последовательный контейнер list?
10. Что означает имя iterator в области видимости последовательного контейнера?
11. Что такое функциональный объект?
12. Назначение и параметры алгоритма sort() для последовательных контейнеров?
13. Назначение и параметры алгоритма accumulate () для последовательных контейнеров?
14. Назначение метода size() последовательного контейнера?
15. В чём особенности статических методов?
16. В чём особенности последовательных контейнеров?

### Содержание отчета

1. Задание.
2. Текст программы.
3. Скриншот.

## Литература

1. Написание модульных тестов для C/C++ в Visual Studio [Электронный ресурс]  URL: <https://docs.microsoft.com/ru-ru/visualstudio/test/writing-unit-tests-for-c-cpp> (дата обращения 21.03.18).
2. Руководство по программированию на C# [Электронный ресурс]  URL: https://metanit.com/cpp/tutorial/1.1.php (дата обращения 20.03.18).